SYSTEM FOR GENERATING OPTIMIZED COMPUTER DATA FIELD CONVERSION ROUTINES

FIELD OF THE INVENTION

The present invention is directed to computer data. More particularly, the present invention is directed to the conversion of one type of computer data field to another type.

BACKGROUND OF THE INVENTION

In many instances during computer processing of information, computer data must be converted from one data field type to another. For example, whenever data is passed from one program to another, the data typically goes though several conversions during the process, such as converting from text digits to a binary number.

EXPRESS MAIL LABEL NO. EI827994554US
DATE OF DEPOSIT: March 19, 1999

The typical technique for converting data includes using a generic data conversion routine. When an entire record of data must be converted, the conversion routine must determine what the characteristics or attributes are for each of the data fields in the record. This may require the conversion routine to execute the same decision tree for each field for each record even though each field has known characteristics that do not change on a row by row basis. Therefore, many computer cycles are wasted by asking questions such as "Is this field of type character, integer, etc.?" over and over for each data field.

Based on the foregoing, there is a need for a system that provides efficient conversion of data fields.

SUMMARY OF THE INVENTION

One embodiment of the present invention is a system for converting data from input field types to output field types. The system receives a plurality of input attributes and output attributes from an application program, dynamically generates a plurality of data field conversion routines for each set of input attributes and output attributes, and stores the plurality of data field conversion routines in memory that is accessible to the application program.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram that illustrates an overview of the functionality of an optimized conversion generator system in accordance with one embodiment of the present invention.

Fig. 2 is a flowchart of the steps performed by the system in accordance with one embodiment of the present invention to generate optimized conversion routines.

Fig. 3 is a flowchart of the steps executed by the application when using the routines to convert input fields to output fields.

Fig. 4 is a flowchart of the code generating steps executed the conversion generator system to generate code when called by the application.

Figs. 5a and 5b illustrate a general example of dynamic code building that is used in one embodiment of the present invention.

Figs. 6a - 6h illustrate a specific example of a dynamic code generation routine that performs CHARACTER to CHARACTER conversions.

DETAILED DESCRIPTION

One embodiment of the present invention is a system that generates optimized data field to data field conversion routines for each type of conversion required by an application program. Fig. 1 is a block diagram that illustrates an overview of the functionality of an optimized conversion generator system 20 in accordance with one

embodiment of the present invention. System 20 can be implemented in software and executed on a general purpose computer that includes a central processing unit, and memory. In one embodiment, system 20 is implemented with IBM/360 machine instructions.

An application program 10 requires one or more types of field conversions to be executed. For each type of conversion, application 10 provides to system 20 the input (or "source") and output (or "destination") field attributes. For each set of input and output field attributes, system 20 dynamically generates an optimized conversion routine 30 that performs the conversion. The optimized routines 30 are placed in storage that is available to application 10.

The routines 30 in one embodiment are generated as stand-alone routines that are capable of being serially reusable and are called by application 10 using, for example, an application program interface ("API") when a conversion is required. In another embodiment, the routines 30 are generated as code chunks that are inserted inline with application 10 and are directly accessed when a conversion is required.

One benefit of the present invention is that by building optimized conversion routines specifically tailored to the input and output field attributes, every execution of the routine saves numerous instructions that would normally be needed to identify field attributes each time the conversion is executed.

Fig. 2 is a flowchart of the steps performed by system 20 in accordance with one embodiment of the present invention to generate optimized conversion routines 30. The steps are executed by system 20 after application 10 determines at step 100 what attributes the input fields and output fields have.

At step 102, system 20 sets up the default process options of the generated conversion routines 30. The options may include whether the generated conversion routines 30 will be callable functions (i.e., able to be called by application 10), or copied inline into application 10. Step 102 builds a template interface block 104 which is an interface between application 10 and conversion generator system 20. Step 102 also generates an initiation call 106 that obtains the necessary storage and checks for errors.

At step 108, a loop is initiated. The loop continues until all fields that must be converted are exhausted.

Within the loop, at step 110 each set of input and output field attributes is received from application 10. The attributes are received through an API, and step 110 also builds a common field conversion interface block 116 based on the attributes.

At step 112, the code generator of system 20 is called, using the common interface block 116. Step 112 generates code 118.

At step 114, a function pointer that points to the generated field conversion routine 30 is saved.

Fig. 3 is a flowchart of the steps executed by application 10 when using routines 30 to convert input fields to output fields.

During step 122, the application is processing. At step 124, the application obtains source or input data to convert. Typically, step 124 involves reading one or more records.

At step 126, a loop is initiated for each record read. At step 128, in one embodiment the appropriate conversion routine 30 for the conversion is called.

When all the data field and records are converted, at step 132 the code generator system 20 is called for termination. This results in freeing up memory at step 134.

At step 136, application 10 continues to process. Finally, at step 138 application 10 is completed.

Fig. 4 is a flowchart of the code generating steps executed by conversion generator system 20 to generate code when called by application 10.

At step 200, system 20 initializes by, for example, establishing the required storage, checking for invalid options, and specifying how the code should be generated.

At step 202, system 20 validates specific field conversion options such as verifying that the input and output lengths are correct. Step 202 also determines how

big the code will be when generated. This can be used by application 10 if the generated code will be stored inline.

At step 204, system 20 builds the conversion routine using field conversion interface block 116.

At step 206, the storage obtained at step 200 is released.

Steps 202 and 204 go through the same internal process. Therefore, at step 208 the input field type is determined. Examples of input field types include character input 210 or special time format input 212. However, any input field type is supported by the present invention.

Similarly, at step 214 the output field type is determined. Examples of output field types also include character input 213 or special time format input 215, but any output field type is supported by the present invention.

At step 216, if step 202 was executed, the size of the generated code is determined. At step 218, if step 204 was executed, the field conversion routines 30 are generated.

As disclosed, system 20 in accordance with one embodiment of the present invention dynamically generates optimized conversion routines 30 for each set of input and output field attributes. Routines 30 are then utilized by application 10 to process conversions. Input and output fields are categorized into archetypal data types by system 20, each with definable attributes and conversion behaviors. For example:

- Character data types will be a fixed length field with a maximum length attribute and a CCSID (or character set code page) attribute.
- Date data types will be a fixed length field with a maximum length attribute and a format attribute (ISO, EUR, etc) which determines location and type of separators used in date.

Some previously described or additional features included in one embodiment of optimized conversion generator system 20 include:

- Optionally obtain and free storage for API control blocks and/or generated code.
- API control blocks can be chained and templated by API management functions.
- API control blocks can be built through use of a macro interface.
- Conversion routines can utilize registers to address the input and output field locations directly. The registers can be chosen by application 10 through API parameters.
- The source field address register may optionally be incremented to the end of the input field after conversion based on API parameters.
- The target field address register may optionally be incremented to the end of the formatted field after conversion based on API parameters.

- An additional register may be incremented by the length of the converted field based on API parameters.
- Standard Linkage may be generated for conversion routines based on API parameters.
- Conversion Error exits may be specified to handle enumerated conversion error conditions based on API parameters.
- Character Code Set translation conversion code can be generated based on API parameters (i.e., ASCII character fields can be translated to EBCDIC character fields).
- Conversion routines can be generated to utilize the latest instructions supported by the level of the operating system for which the code is being generated.

In one embodiment, system 20 dynamically generates code by building code chunks in storage accessible by calling application 10 based on various settings in the API control block. Generating the code involves the following steps, as discussed in conjunction with the flowcharts:

1. Obtain storage for the code.
2. Identify code templates needed.
3. Move code templates.
4. Modify code templates.

5. Return executable code to calling application.

Further, in one embodiment system 20 can optionally, based on the API specification, generate program debugging instrumentation for the dynamically generated code. This instrumentation can include an optional dynamically allocated output file containing, for each field conversion: a report of the API options used for each dynamically generated routine that can be used to insure correctness of field attributes and general processing options; and a disassembled listing of the dynamically generated routine provided by an internal disassembler within system 20 that can be used to identify conversion code inaccuracies and areas of further optimization, and to help resolve generated code failures.

Figs. 5a and 5b illustrate a general example of dynamic code building that is used in one embodiment of the present invention.

Figs. 6a - 6h illustrate a specific example of a dynamic code generation routine that performs CHARACTER to CHARACTER conversions.

Several embodiments of the present invention are specifically illustrated and/or described herein. However, it will be appreciated that modifications and variations of the present invention are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.